Chapter XVII
MDD Approach for Maintaining Integrity Constraints in Databases

Harith T. Al-Jumaily
Carlos III University of Madrid, Spain

Dolores Cuadra
Carlos III University of Madrid, Spain

Paloma Martínez
Carlos III University of Madrid, Spain

INTRODUCTION

In the context of database, we believe that **MDD** (Model-Driven Development) (OMG, 2006) is a very ambitious task because we find that when applying database development methodologies such as (Elmasri, et al., 2007), there are processes devoted to transforming conceptual into logical schemata. In such processes, semantic losses are produced since logical elements are not coincident with conceptual elements. A correct constraints transformation is necessary to preserve the semantics that reflects the Universe of Discourse. The **multiplicity constraint,** also called cardinality constraint, is one of these constraints that can be established in a conceptual schema. It has dynamic aspects that are transformed into the logical model as certain conditions to verify the insertion, deletion, and update operations. The verification of these constraints is a serious and complex problem because currently database systems are not able to preserve the **multiplicity constraints** of their objects. To solve the modeling problem, **CASE tools** have been introduced to automate the life cycle of database development. These platforms try to help the database developers in different design phases. Nevertheless, these tools are frequently simple graphical interfaces and do not completely carryout the design methodology that they are should to support.

Therefore, in this work the **MDD** approach has been considered to enhance the **transformation**

rules of the conceptual schema into the relational schema. The relational model was considered in this work because most database methodologies are agreeing with it to transform the conceptual schema into a logical one. A tool was plugged into Rational Rose to ensure this task. This tool can automatically generate maintaining mechanisms for these constraints to a target DBMS; triggers system as maintaining mechanisms is used. These mechanisms can provide a fundamental base to obtain a very high level of knowledge independence (Paton, 1999). The triggers system is specified according to the recent **SQL:2003 standard** that revises all parts of SQL99 and adds new features (ISO Standard 2003).

Because triggers development is more complicated than traditional method, we have detected that generating triggers and plugging those into a given schema is an insufficient task because the behaviour of triggers is not clear; they could produce cycles in the execution and needs to be verified. Therefore, besides the transformation of integrity constraints into triggers, our plugged-in tool builds UML sequence diagrams to verify the interaction of these triggers with themselves first, then with the other elements in the schema. These contributions make our approach quite useful, practical, and intuitive to manage triggers.

The rest of this chapter is organized as the following. Related works are presented in the next section. Therefore, the adaptation of **MDD** to build our tool is discussed. In the Add-in Module Design section, we discuss how the integration of the active technology into UML schema is performed. And finally, some conclusions and future works are presented.

RELATED WORKS

When the active technology was introduced into database systems, the automatic transformation from constraints specification to **active rules** has been well considered in the literature. Different approaches were used to transform integrity constraints into **active rules**. Some of these approaches reject updates when the violation occurs, and the initial state before updates is restored (Decker, 2006). Other approach in which inconsistency states are detected first, but consistency is restored by issuing corrective actions that depend on the particular constraint violation (Ceri et al., 1997).

In some works, such as (Ceri et al., 1990) is described a general framework to transform constraints into production **active rules** for constraint maintaining. They define a general language for expressing integrity constraints, and **transformation rules** are used to convert integrity constraints into **active rules**. The contribution in (Türker et al., 2000) is very important to our work because they issued some simple rules that are independent of a particular commercial system. These rules are used for implementing triggers based on constraints specifications by using Tuple Relational Calculus (Elmasri, et al., 2007). In this work, we will specify the **transformation rules** based on constraints specifications by using **OCL** (Object Constraints Language). Our approach agrees with the proposal in (Olivé, 2003) that introduced **OCL** as a method to provide the definition of integrity constraints in conceptual schemas as constraint operations. In this proposal, constraints have been introduced in the conceptual schema as operations without defining any rules to specify the transformation of such operations to a logical schema.

On the other hand, working with **active rules**/triggers needs to ensure the execution termination. The verification of **active rules**/triggers execution is the major problem that makes applications development a difficult task. Because of **active rules** can act in such ways that lead to conflict and undesirable executions, database developers need additional effort to control rules behavior. The objective of this verification is to guarantee the termination of triggers execution.

Termination means that the execution of any set of **active rules** must terminate. This needs to